MEM 591 – Homework 5

Khac Hieu Dinh

Problem 1:

a. Using fzero() to find ALL zeros of f(x) in the interval:

Since fzero(), like other numerical root-finding algorithms, will only produce one guess given a search interval, we have sub-divide the interval from -2 to 2 into multiple smaller intervals to ensure that no zero is missed (if two zeros are in the same search interval, one of them will be missed). Given that the provided function $\sin(20x) - x$ is continuous, only intervals where the function change sign will be checked (fzero() will raise an error if the function have the same sign when evaluated at the end points).

Given the angular frequency of the sine wave in the equation (20 rad per unit of x, corresponding to a period of approximately 0.3 unit of x), a safe number of intervals to use is 51, since $\frac{4}{50} = 0.08$, which is much smaller than the sine wave's frequency.

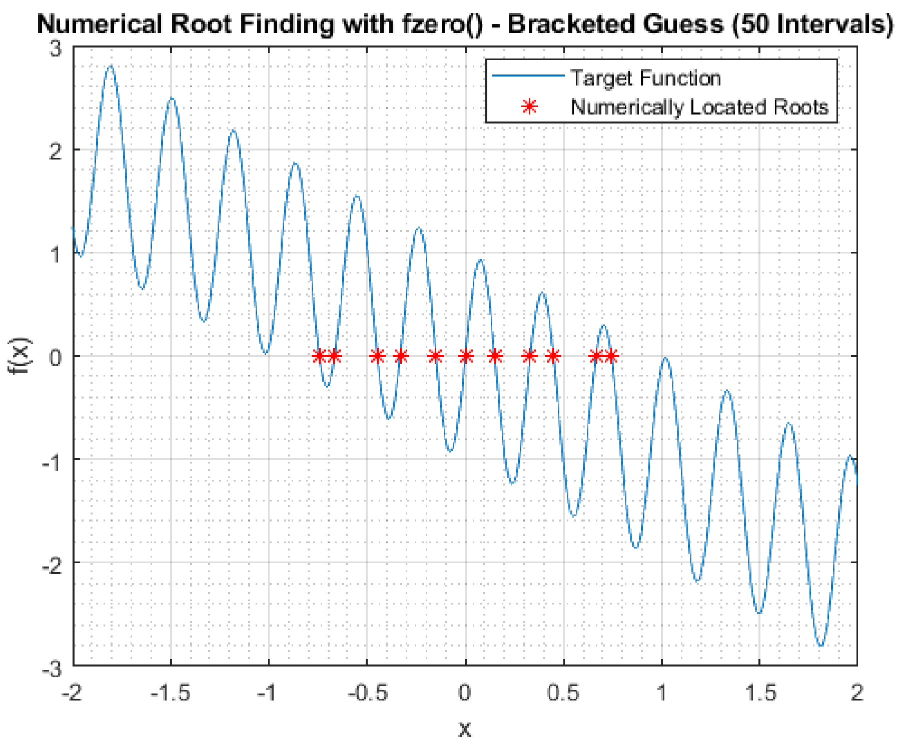
Random perturbations of 2 orders of magnitude less than the interval width is also added to the intervals to reduce the probability of the sub-bracket endpoints being the root themselves. For example, since 0 is a root AND a nodal points (if we use 50 equally spaced search interval), applying the procedure above without random nodal perturbation would result in the root x = 0 being missed. This is normally not needed for real equations with non-round parameters, but for the purpose of this exercise, it is required).

Note that since the sub-intervals do not overlap, the root found inside two different intervals cannot be the same.

```
clc; clear; close all
TOL = 1e-6;
MAX = 2;
MIN = -2;
NUM_NODE = 51;
interval = (MAX - MIN)/(NUM_NODE - 1);
f_x = @(x) sin(20*x) - x;

node_series = linspace(MIN,MAX,NUM_NODE);
node_series(2:end-1) = node_series(2:end-1) + interval*0.01*(rand(1, NUM_NODE - 2) -
0.5); % Add randomness to the end points, except for the original bound
f_series = f_x(node_series);

plot_x_series = linspace(MIN,MAX,1000); % Different, smoother series just for plotting
plot_f_series = f_x(plot_x_series);

root_guess = NaN(NUM_NODE - 1,1); % Num bracket = num node - 1

for index = 1:(NUM_NODE - 1)
```

```
    if f_series(index)*f_series(index+1) < 0 % If the sign of the end points is the
same, fzero will fail.
        root_guess(index) = fzero(f_x, [node_series(index) node_series(index+1)]);
    end
end

root_guess = root_guess(~isnan(root_guess)) % Using logical indexing to
% eliminate all invalid brackets (NaN root guess = bracket with same sign)
% Note that there will be no duplicated roots, since the brackets does not
% overlap

figure;
plot(plot_x_series, plot_f_series, "DisplayName","Target Function"); hold on
plot(root_guess,zeros(length(root_guess),1),"r*", "DisplayName", "Numerically Located
Roots");
title(sprintf("Numerical Root Finding with fzero() - Bracketed Guess (%d
Intervals)",NUM_NODE -1));
xlabel("x");
ylabel("f(x)");
legend(Location="best");
grid on; grid minor;
```

```
root_guess =

   -0.7435
   -0.6647
   -0.4480
   -0.3310
   -0.1496
   -0.0000
    0.1496
    0.3310
    0.4480
    0.6647
    0.7435
```



It can be seen that all 11 roots were found by the algorithm used.

Another simpler approach (that require human judgement) is to plot the function first and visually locate the brackets that contains an unique root and then refine those locations using fzero(). However, in the interest of autonomous function solving, the presented subdivision and seek method is used instead.

b. Implementation of bisection search method:

For this subroutine, the criteria that will be used for stopping the search is the provided error tolerance (i.e., the desired interval length that contains the root).

Of course, the subroutine will also require that the function have different signs when evaluated at end points of the provided interval.

The intermediate iterations are logged in a table and will be printed out for debugging purposes.

Subroutine

```
function x_mid = myBisection(func, x_start, x_end, tol)

if x_start >= x_end
    error("Invalid Interval: End Points Ordering Violated")
end

f_start = func(x_start);
f_end = func(x_end);

if f_start*f_end >= 0
    error("Invalid Interval: Function at End Points Should Have Different Sign");
end
% Number of steps can be pre-calculated for bisection algorithm
num_iter = ceil(log2((x_end -x_start)/tol)); % ALways round up, otherwise tolerance
will not be met (1 step short)

% Debug table allocation
iter_index_list = (0:num_iter)';
x_start_list = zeros(num_iter + 1,1);
f_start_list = zeros(num_iter + 1,1);
x_end_list = zeros(num_iter + 1,1);
f_end_list = zeros(num_iter + 1,1);
e_list = zeros(num_iter + 1,1);

x_mid = (x_start + x_end) / 2;
f_mid = func(x_mid);

x_start_list(1) = x_start;
x_end_list(1) = x_end;
e_list(1) = x_end - x_start;

for iter = 1:num_iter
```

```
    if f_mid*f_start < 0
        x_end = x_mid;
    else
        x_start = x_mid;
    end

    % Logging
    x_start_list(iter + 1) = x_start;
    f_start_list(iter + 1) = f_start;
    x_end_list(iter + 1) = x_end;
    f_end_list(iter + 1) = f_end;
    e_list(iter + 1) = x_end - x_start;

    x_mid = (x_start + x_end)/2;
    f_start = func(x_start);
    f_end = func(x_end);
    f_mid = func(x_mid);

end

debug_table = table(iter_index_list, x_start_list, f_start_list, x_end_list,
f_end_list, e_list);

disp(debug_table)

end
```

Results

```
clc; clear; close all;

x_start = 0.5;
x_end = 1.5;
tol = 1e-6;
func = @(x) x*sin(x) - 0.5;

root = myBisection(func,x_start, x_end, tol)

should_be_zero = func(root)
```

iter_index_list	x_start_list	f_start_list	x_end_list	f_end_list	e_list
0	0.5	0	1.5	0	1
1	0.5	-0.26029	1	0.99624	0.5
2	0.5	-0.26029	0.75	0.34147	0.25
3	0.625	-0.26029	0.75	0.011229	0.125
4	0.6875	-0.13431	0.75	0.011229	0.0625
5	0.71875	-0.063708	0.75	0.011229	0.03125
6	0.73438	-0.026743	0.75	0.011229	0.015625
7	0.73438	-0.0078781	0.74219	0.011229	0.0078125
8	0.73828	-0.0078781	0.74219	0.0016458	0.0039062
9	0.74023	-0.0031237	0.74219	0.0016458	0.0019531
10	0.74023	-0.0007408	0.74121	0.0016458	0.00097656
11	0.74072	-0.0007408	0.74121	0.00045203	0.00048828
12	0.74072	-0.0001445	0.74097	0.00045203	0.00024414
13	0.74072	-0.0001445	0.74084	0.00015373	0.00012207
14	0.74078	-0.0001445	0.74084	4.6071e-06	6.1035e-05
15	0.74081	-6.995e-05	0.74084	4.6071e-06	3.0518e-05
16	0.74083	-3.2672e-05	0.74084	4.6071e-06	1.5259e-05
17	0.74084	-1.4033e-05	0.74084	4.6071e-06	7.6294e-06

```
          18              0.74084       -4.7127e-06       0.74084       4.6071e-06    3.8147e-06
          19              0.74084       -5.2809e-08       0.74084       4.6071e-06    1.9073e-06
          20              0.74084       -5.2809e-08       0.74084       2.2772e-06    9.5367e-07


root =

    0.7408


should_be_zero =

   5.2968e-07
```

Problem 2:

Before attempting to find the roots of polynomials, it is important to have some visual idea of where they are (so that the appropriate initial guess can be chosen).

(Note that the plot is very zoomed in. If both axes are scaled the same, the function look very "flat")

In the plots, there are 4 visible roots, all very close to 0. This:

The sequence of initial guess values for Newton method: $x_0 = 0.0, 0.2, 0.5, 0.7$

The sequence of initial guess values for secant method: $(x_0, x_1) = (0.0, 0.01), (0.2, 0.21), (0.5, 0.49), (0.7, 0.69)$

a. Newton Method

Termination Condition: Relative Approximate Error Reaches a Certain Threshold or 30 Iteration is Reached.

Subroutine

```
function guess = myNewtonRoot(func, deri, guess, rel_tol)

MAX_ITER = 30;
ZERO_TOL = 1e-8;

iter_index = (1:MAX_ITER)';
guess_list = zeros(MAX_ITER,1);
next_guess_list = zeros(MAX_ITER,1);
f_list = zeros(MAX_ITER, 1);
deri_list = zeros(MAX_ITER, 1);
e_approx_list = zeros(MAX_ITER, 1);
r_list = zeros(MAX_ITER, 1);


prev_guess = NaN;

for iter = 1:30

    current_deri = deri(guess);
```

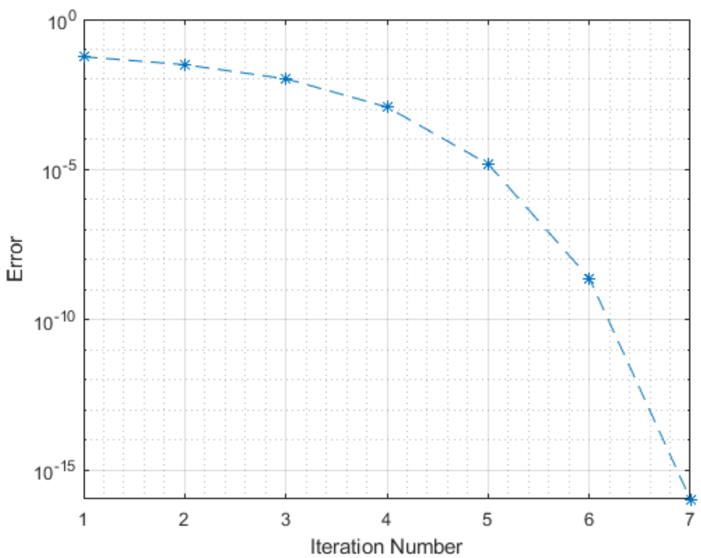
```
    if abs(current_deri) < ZERO_TOL
        error("ERROR: Iterating Near Flat Region. Solution Instability.")
    end

    current_f = func(guess);

    prev_guess = guess;
    guess = prev_guess - current_f/current_deri;

    e = abs(guess - prev_guess);
    rel_e = e/abs(guess);

    % Logging
    guess_list(iter) = prev_guess;
    next_guess_list(iter) = guess;
    f_list(iter) = current_f;
    deri_list(iter) = current_deri;
    e_approx_list(iter) = e;
    r_list(iter) = abs(guess/prev_guess);

    if rel_e <= rel_tol
        debug_table = table( ...
            iter_index(1:iter), ...
            guess_list(1:iter), ...
            f_list(1:iter), ...
            deri_list(1:iter), ...
            next_guess_list(1:iter),...
            e_approx_list(1:iter), ...
            r_list(1:iter), ...
            VariableNames = ["n", "x_n","f(x)", "df/dx(x)", "x_{n+1}","h = |x_{n+1} -
x_n|", "r = |x_{n+1}/x_n|"]);
        disp(debug_table);
        figure;
        semilogy(iter_index(1:iter),e_approx_list(1:iter),"--*")
        xlabel("Iteration Number");
        ylabel("Error");
        grid on, grid minor;

        return
    end

end

error("ERROR: Failure to Converge After %d Iterations.", MAX_ITER);

end
```

Results

```
root1 = myNewtonRoot(f,df,guess1,rel_tol)
```

```
    n      x_n          f(x)         df/dx(x)      x_{n+1}      h = |x_{n+1} - x_n|     r = |x_{n+1}/x_n|
    _    ________    __________    _________    ________    ___________________    _________________

    1           0        0.0072       -0.126    0.057143             0.057143                  Inf
    2    0.057143     0.0019372    -0.062397    0.088189             0.031046               1.5433
    3    0.088189    0.00039923    -0.037748    0.098766             0.010576               1.1199
    4    0.098766    3.7506e-05     -0.03077    0.099985            0.0012189               1.0123
    5    0.099985    4.6429e-07     -0.03001         0.1           1.5471e-05               1.0002
    6         0.1     7.421e-11        -0.03         0.1           2.4737e-09                    1
    7         0.1    3.4694e-18        -0.03         0.1           1.1102e-16                    1
```

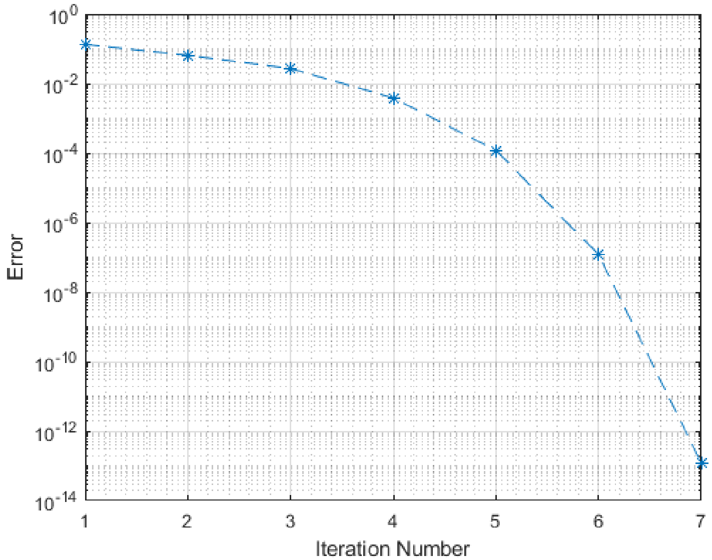
```
root1 =

    0.1000
```




```
root2 = myNewtonRoot(f,df,guess2,rel_tol)
```

```
    n      x_n          f(x)          df/dx(x)      x_{n+1}      h = |x_{n+1} - x_n|     r = |x_{n+1}/x_n|
    _    _______    ___________    _________    _______    ___________________    _________________

    1        0.2        -0.0008        0.006    0.33333              0.13333               1.6667
    2    0.33333     0.00013827    0.0021481    0.26897             0.064368               0.8069
    3    0.26897    -0.00022746     0.008406    0.29602             0.027059               1.1006
    4    0.29602     -2.463e-05    0.0063878    0.29988            0.0038558                1.013
    5    0.29988    -7.1865e-07     0.006012        0.3           0.00011954               1.0004
    6        0.3    -7.1377e-10        0.006        0.3           1.1896e-07                    1
    7        0.3    -7.1991e-16        0.006        0.3           1.1996e-13                    1
```

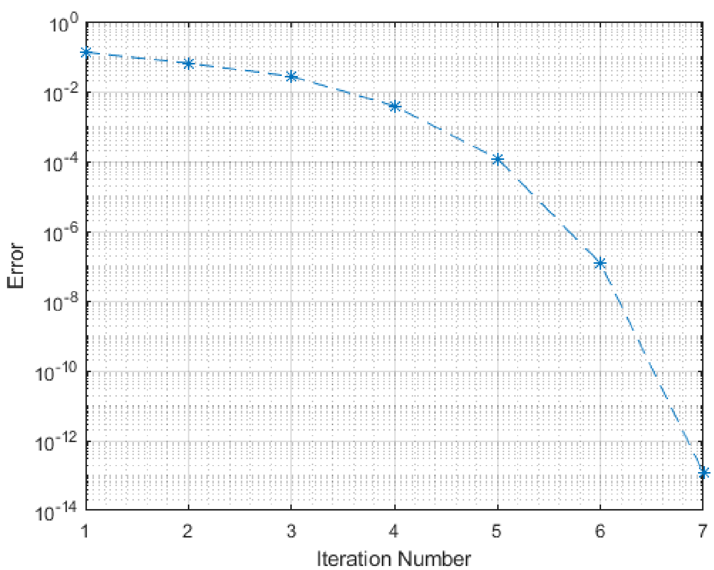
```
root2 =

    0.3000
```




```
root3 = myNewtonRoot(f,df,guess3,rel_tol)
```

n	x_n	f(x)	df/dx(x)	x_{n+1}	h = |x_{n+1} - x_n|	r = |x_{n+1}/x_n|
_	_______	___________	__________	_______	___________________	_________________
1	0.5	-0.0008	-0.006	0.36667	0.13333	0.73333
2	0.36667	0.00013827	-0.0021481	0.43103	0.064368	1.1755
3	0.43103	-0.00022746	-0.008406	0.40398	0.027059	0.93722
4	0.40398	-2.463e-05	-0.0063878	0.40012	0.0038558	0.99046
5	0.40012	-7.1865e-07	-0.006012	0.4	0.00011954	0.9997
6	0.4	-7.1377e-10	-0.006	0.4	1.1896e-07	1
7	0.4	-7.1297e-16	-0.006	0.4	1.1885e-13	1

```
root3 =

    0.4000
```

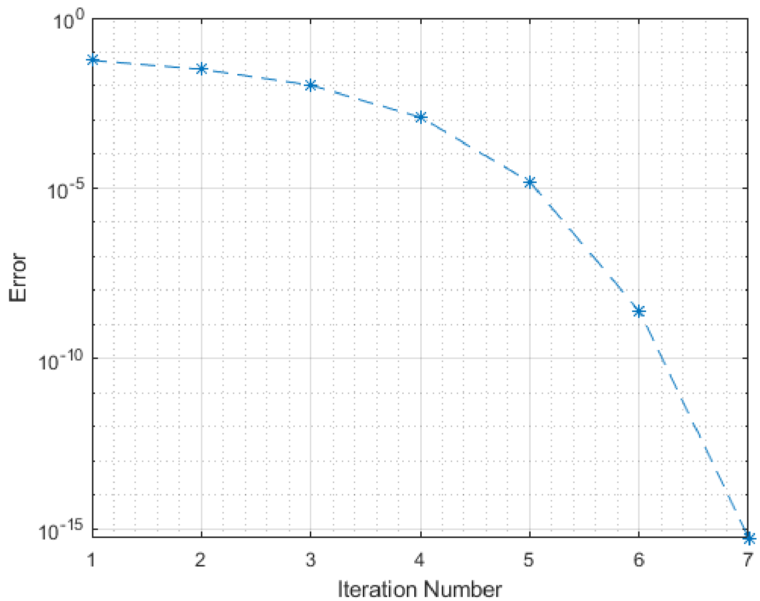



```
root4 = myNewtonRoot(f,df,guess4,rel_tol)
```

n x_n f(x) df/dx(x) x_{n+1} h = |x_{n+1} - x_n| r = |x_{n+1}/x_n|

_	_______	__________	________	_______	___________________	_________________
1	0.7	0.0072	0.126	0.64286	0.057143	0.91837
2	0.64286	0.0019372	0.062397	0.61181	0.031046	0.95171
3	0.61181	0.00039923	0.037748	0.60123	0.010576	0.98271
4	0.60123	3.7506e-05	0.03077	0.60002	0.0012189	0.99797
5	0.60002	4.6429e-07	0.03001	0.6	1.5471e-05	0.99997
6	0.6	7.421e-11	0.03	0.6	2.4737e-09	1
7	0.6	1.5613e-17	0.03	0.6	5.5511e-16	1

```
root4 =

    0.6000
```



Comment

Based on theoretical analysis, the convergence rate for Newton method (non-duplicate root) is **quadratic**. Indeed, by observing the error vs iteration number for each plot, we can see that the order of magnitudes number of the error (the negative power) roughly double after each iteration (after an initial set of iterations). For example, for the last root, the order of magnitude sequence is: $10^{-3}, 10^{-5}, 10^{-8}, 10^{-15}$ (r is close to 2, but not quite).

b. Secant Method

Termination Condition: Relative Approximate Error Reaches a Certain Threshold or 30 Iteration is Reached.

Subroutine

```
function next_guess = mySecantRoot(func, x0, x1, rel_tol)

MAX_ITER = 30;
ZERO_TOL = 1e-8;

iter_index = (1:MAX_ITER)';
guess_0_list = zeros(MAX_ITER,1);
guess_1_list = zeros(MAX_ITER,1);
next_guess_list = zeros(MAX_ITER,1);
f_list = zeros(MAX_ITER, 1);
deri_list = zeros(MAX_ITER, 1);
e_approx_list = zeros(MAX_ITER, 1);
r_list = zeros(MAX_ITER, 1);


prev_guess_0 = x0;
prev_guess_1 = x1;

for iter = 1:30

    current_deri = (func(prev_guess_1) - func(prev_guess_0)) / (prev_guess_1 -
prev_guess_0);

    if abs(current_deri) < ZERO_TOL
        error("ERROR: Iterating Near Flat Region. Solution Instability.")
    end

    current_f = func(prev_guess_1);

    next_guess = prev_guess_1 - current_f/current_deri;

    e = abs(next_guess - prev_guess_1);
    rel_e = e/abs(next_guess);

    % Logging
    guess_0_list(iter) = prev_guess_0;
    guess_1_list(iter) = prev_guess_1;
    next_guess_list(iter) = next_guess;
    f_list(iter) = current_f;
    deri_list(iter) = current_deri;
    e_approx_list(iter) = e;
    r_list(iter) = abs(next_guess/prev_guess_1);

    if rel_e <= rel_tol
        debug_table = table( ...
            iter_index(1:iter), ...
            guess_0_list(1:iter), ...
            guess_1_list(1:iter), ...
```

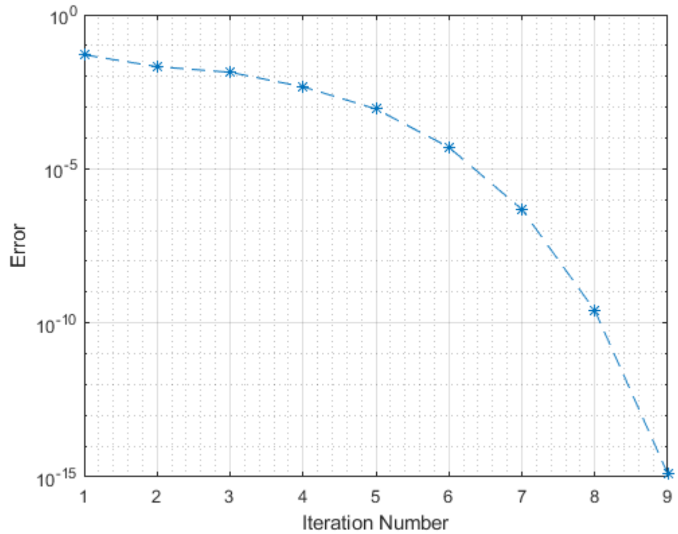
```
            f_list(1:iter), ...
            deri_list(1:iter), ...
            next_guess_list(1:iter),...
            e_approx_list(1:iter), ...
            r_list(1:iter), ...
            VariableNames = ["n", "x_n","x{n-1}","f(x)", "df/dx(x)", "x_{n+1}","h =
|x_{n+1} - x_n|", "r = |x_{n+1}/x_n|"]);
        disp(debug_table);
        figure;
        semilogy(iter_index(1:iter),e_approx_list(1:iter),"--*")
        xlabel("Iteration Number");
        ylabel("Error");
        grid on, grid minor;

        return
    end

    prev_guess_0 = prev_guess_1;
    prev_guess_1 = next_guess;

end

error("ERROR: Failure to Converge After %d Iterations.", MAX_ITER);

end
```

Results

```
root1_sec = mySecantRoot(f,guess1_0, guess1_1, rel_tol)
```

n	x_n	x{n-1}	f(x)	df/dx(x)	x_{n+1}	h = |x_{n+1} - x_n|	r = |x_{n+1}/x_n|
1	0	0.01	0.0060056	-0.11944	0.060282	0.050282	6.0282
2	0.01	0.060282	0.0017457	-0.08472	0.080888	0.020606	1.3418
3	0.060282	0.080888	0.00069372	-0.051054	0.094476	0.013588	1.168
4	0.080888	0.094476	0.00017536	-0.038148	0.099072	0.0045967	1.0487
5	0.094476	0.099072	2.8094e-05	-0.032037	0.099949	0.00087693	1.0089
6	0.099072	0.099949	1.5194e-06	-0.030304	0.1	5.0138e-05	1.0005
7	0.099949	0.1	1.4455e-08	-0.030016	0.1	4.8157e-07	1
8	0.1	0.1	7.5579e-12	-0.03	0.1	2.5193e-10	1
9	0.1	0.1	3.8164e-17	-0.03	0.1	1.2768e-15	1

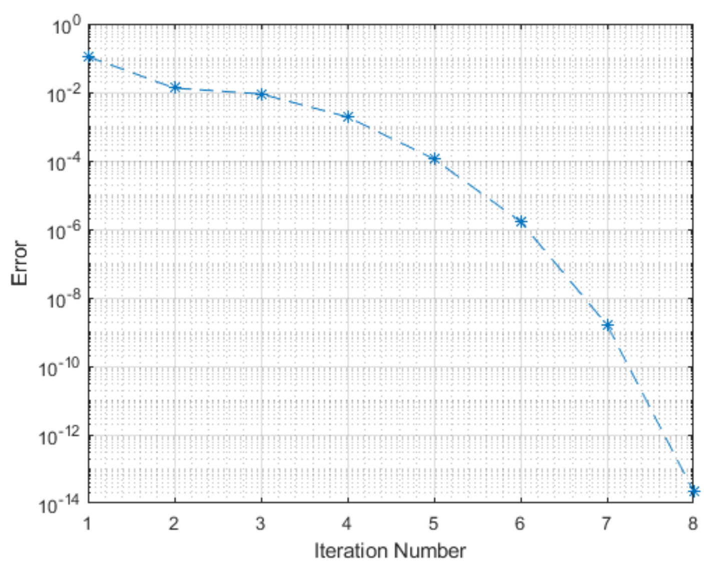
```
root1_sec =

    0.1000
```




```
root2_sec = mySecantRoot(f,guess2_0, guess2_1, rel_tol)
```

n	x_n	x{n-1}	f(x)	df/dx(x)	x_{n+1}	h = |x_{n+1} - x_n|	r = |x_{n+1}/x_n|
1	0.2	0.21	-0.00073359	0.006641	0.32046	0.11046	1.526
2	0.21	0.32046	0.00010031	0.007549	0.30718	0.013287	0.95854
3	0.32046	0.30718	4.0413e-05	0.0045076	0.29821	0.0089656	0.97081
4	0.30718	0.29821	-1.0893e-05	0.0057225	0.30011	0.0019035	1.0064
5	0.29821	0.30011	6.8637e-07	0.0060831	0.3	0.00011283	0.99962
6	0.30011	0.3	1.0034e-08	0.0059942	0.3	1.674e-06	0.99999
7	0.3	0.3	-9.5885e-12	0.0059999	0.3	1.5981e-09	1
8	0.3	0.3	1.3357e-16	0.006	0.3	2.226e-14	1

```
root2_sec =

    0.3000
```

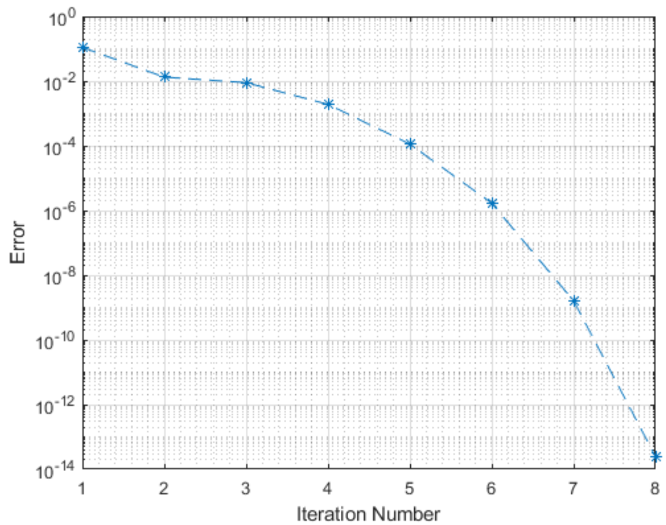



```
root3_sec = mySecantRoot(f,guess3_0, guess3_1, rel_tol)
```

n	x_n	x{n-1}	f(x)	df/dx(x)	x_{n+1}	h = |x_{n+1} - x_n|	r = |x_{n+1}/x_n|

1	0.5	0.49	-0.00073359	-0.006641	0.37954	0.11046	0.77456
2	0.49	0.37954	0.00010031	-0.007549	0.39282	0.013287	1.035
3	0.37954	0.39282	4.0413e-05	-0.0045076	0.40179	0.0089656	1.0228
4	0.39282	0.40179	-1.0893e-05	-0.0057225	0.39989	0.0019035	0.99526
5	0.40179	0.39989	6.8637e-07	-0.0060831	0.4	0.00011283	1.0003
6	0.39989	0.4	1.0034e-08	-0.0059942	0.4	1.674e-06	1
7	0.4	0.4	-9.5886e-12	-0.0059999	0.4	1.5981e-09	1
8	0.4	0.4	1.4745e-16	-0.006	0.4	2.4591e-14	1

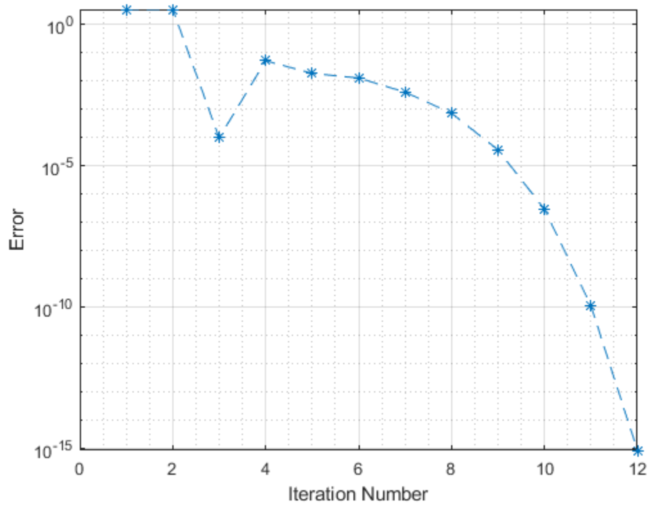
```
root3_sec =

    0.4000
```




```
root4_sec = mySecantRoot(f,guess1_0, guess4_1, rel_tol)
```

n	x_n	x{n-1}	f(x)	df/dx(x)	x_{n+1}	h = |x_{n+1} - x_n|	r = |x_{n+1}/x_n|
1	0	0.69	0.0060056	-0.001731	4.1594	3.4694	6.0282
2	0.69	4.1594	209.65	60.426	0.6899	3.4695	0.16586
3	4.1594	0.6899	0.0059944	60.424	0.6898	9.9205e-05	0.99986
4	0.6899	0.6898	0.0059832	0.11283	0.63677	0.053029	0.92312
5	0.6898	0.63677	0.0015739	0.083149	0.61784	0.018929	0.97027
6	0.63677	0.61784	0.00063979	0.049349	0.60488	0.012965	0.97902
7	0.61784	0.60488	0.00015386	0.037481	0.60077	0.0041051	0.99321
8	0.60488	0.60077	2.34e-05	0.031781	0.60004	0.00073631	0.99877
9	0.60077	0.60004	1.1255e-06	0.030252	0.6	3.7203e-05	0.99994
10	0.60004	0.6	8.9443e-09	0.030012	0.6	2.9803e-07	1
11	0.6	0.6	3.4651e-12	0.03	0.6	1.155e-10	1
12	0.6	0.6	-2.6021e-17	0.03	0.6	8.8818e-16	1

```
root4_sec =

    0.6000
```



Comment

The convergence rate of the secant method should be super-linear but not quite quadratic ($r \approx 1.6$). Indeed, by observing the error vs iteration number plot, we can see that the number of significant digits (the negative powers) increases by approximately 50% every iteration. For example, for the last root, the order of magnitude sequence (after iteration 8) is: $10^{-3}, 10^{-4.5}, 10^{-6,5}, 10^{-10}, 10^{-15}$, which is the expected trend.

c. Verification with MATLAB

```
syms x;
f = x^4 - 1.4*x^3 + 0.67*x^2 - 0.126*x + 0.0072;

MATLAB_symbolic_solver = solve(f, x)
```

```
MATLAB_symbolic_solver =

1/10
3/10
 2/5
 3/5
```

All 4 roots are the same as those obtained using numerical method.

Appendix

a. Subroutine

```
function x_mid = myBisection(func, x_start, x_end, tol)

if x_start >= x_end
    error("Invalid Interval: End Points Ordering Violated")
end
```

```
f_start = func(x_start);
f_end = func(x_end);

if f_start*f_end >= 0
    error("Invalid Interval: Function at End Points Should Have Different Sign");
end
% Number of steps can be pre-calculated for bisection algorithm
num_iter = ceil(log2((x_end -x_start)/tol)); % ALways round up, otherwise tolerance
will not be met (1 step short)

% Debug table allocation
iter_index_list = (0:num_iter)';
x_start_list = zeros(num_iter + 1,1);
f_start_list = zeros(num_iter + 1,1);
x_end_list = zeros(num_iter + 1,1);
f_end_list = zeros(num_iter + 1,1);
e_list = zeros(num_iter + 1,1);

x_mid = (x_start + x_end) / 2;
f_mid = func(x_mid);

x_start_list(1) = x_start;
x_end_list(1) = x_end;
e_list(1) = x_end - x_start;

for iter = 1:num_iter

    if f_mid*f_start < 0
        x_end = x_mid;
    else
        x_start = x_mid;
    end

    % Logging
    x_start_list(iter + 1) = x_start;
    f_start_list(iter + 1) = f_start;
    x_end_list(iter + 1) = x_end;
    f_end_list(iter + 1) = f_end;
    e_list(iter + 1) = x_end - x_start;

    x_mid = (x_start + x_end)/2;
    f_start = func(x_start);
    f_end = func(x_end);
    f_mid = func(x_mid);

end

debug_table = table(iter_index_list, x_start_list, f_start_list, x_end_list,
f_end_list, e_list);

disp(debug_table)

end
```

```
function guess = myNewtonRoot(func, deri, guess, rel_tol)

MAX_ITER = 30;
ZERO_TOL = 1e-8;

iter_index = (1:MAX_ITER)';
guess_list = zeros(MAX_ITER,1);
next_guess_list = zeros(MAX_ITER,1);
f_list = zeros(MAX_ITER, 1);
deri_list = zeros(MAX_ITER, 1);
e_approx_list = zeros(MAX_ITER, 1);
r_list = zeros(MAX_ITER, 1);


prev_guess = NaN;

for iter = 1:30

    current_deri = deri(guess);

    if abs(current_deri) < ZERO_TOL
        error("ERROR: Iterating Near Flat Region. Solution Instability.")
    end

    current_f = func(guess);

    prev_guess = guess;
    guess = prev_guess - current_f/current_deri;

    e = abs(guess - prev_guess);
    rel_e = e/abs(guess);

    % Logging
    guess_list(iter) = prev_guess;
    next_guess_list(iter) = guess;
    f_list(iter) = current_f;
    deri_list(iter) = current_deri;
    e_approx_list(iter) = e;
    r_list(iter) = abs(guess/prev_guess);

    if rel_e <= rel_tol
        debug_table = table( ...
            iter_index(1:iter), ...
            guess_list(1:iter), ...
            f_list(1:iter), ...
            deri_list(1:iter), ...
            next_guess_list(1:iter),...
            e_approx_list(1:iter), ...
            r_list(1:iter), ...
            VariableNames = ["n", "x_n","f(x)", "df/dx(x)", "x_{n+1}","h = |x_{n+1} -
x_n|", "r = |x_{n+1}/x_n|"]);
        disp(debug_table);
        figure;
        semilogy(iter_index(1:iter),e_approx_list(1:iter),"--*")
        xlabel("Iteration Number");
```

```
        ylabel("Error");
        grid on, grid minor;

        return
    end

end

error("ERROR: Failure to Converge After %d Iterations.", MAX_ITER);

end
```

```
function next_guess = mySecantRoot(func, x0, x1, rel_tol)

MAX_ITER = 30;
ZERO_TOL = 1e-8;

iter_index = (1:MAX_ITER)';
guess_0_list = zeros(MAX_ITER,1);
guess_1_list = zeros(MAX_ITER,1);
next_guess_list = zeros(MAX_ITER,1);
f_list = zeros(MAX_ITER, 1);
deri_list = zeros(MAX_ITER, 1);
e_approx_list = zeros(MAX_ITER, 1);
r_list = zeros(MAX_ITER, 1);


prev_guess_0 = x0;
prev_guess_1 = x1;

for iter = 1:30

    current_deri = (func(prev_guess_1) - func(prev_guess_0)) / (prev_guess_1 -
prev_guess_0);

    if abs(current_deri) < ZERO_TOL
        error("ERROR: Iterating Near Flat Region. Solution Instability.")
    end

    current_f = func(prev_guess_1);

    next_guess = prev_guess_1 - current_f/current_deri;

    e = abs(next_guess - prev_guess_1);
    rel_e = e/abs(next_guess);

    % Logging
    guess_0_list(iter) = prev_guess_0;
    guess_1_list(iter) = prev_guess_1;
    next_guess_list(iter) = next_guess;
    f_list(iter) = current_f;
    deri_list(iter) = current_deri;
    e_approx_list(iter) = e;
    r_list(iter) = abs(next_guess/prev_guess_1);
```

```
    if rel_e <= rel_tol
        debug_table = table( ...
            iter_index(1:iter), ...
            guess_0_list(1:iter), ...
            guess_1_list(1:iter), ...
            f_list(1:iter), ...
            deri_list(1:iter), ...
            next_guess_list(1:iter),...
            e_approx_list(1:iter), ...
            r_list(1:iter), ...
            VariableNames = ["n", "x_n","x{n-1}","f(x)", "df/dx(x)", "x_{n+1}","h = 
|x_{n+1} - x_n|", "r = |x_{n+1}/x_n|"]);
        disp(debug_table);
        figure;
        semilogy(iter_index(1:iter),e_approx_list(1:iter),"--*")
        xlabel("Iteration Number");
        ylabel("Error");
        grid on, grid minor;

        return
    end

    prev_guess_0 = prev_guess_1;
    prev_guess_1 = next_guess;

end

error("ERROR: Failure to Converge After %d Iterations.", MAX_ITER);

end
```

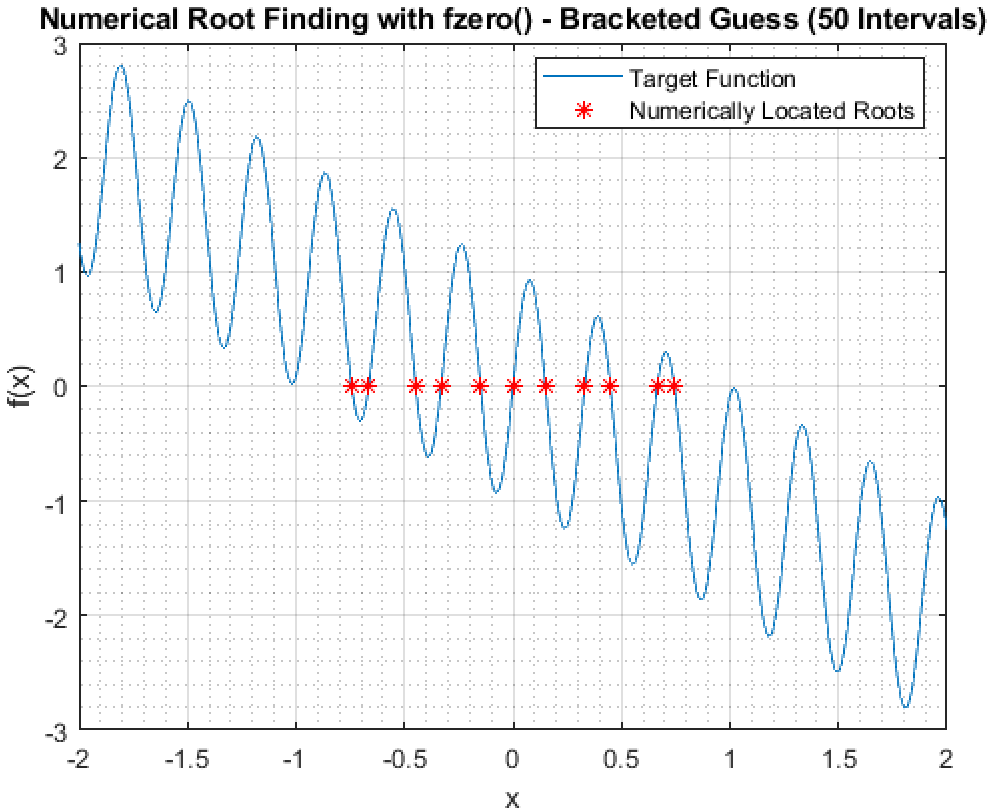
b. Published Scripts

1A

```
clc; clear; close all
TOL = 1e-6;
MAX = 2;
MIN = -2;
NUM_NODE = 51;
interval = (MAX - MIN)/(NUM_NODE - 1);
f_x = @(x) sin(20*x) - x;

node_series = linspace(MIN,MAX,NUM_NODE);
node_series(2:end-1) = node_series(2:end-1) + interval*0.01*(rand(1, NUM_NODE - 2) -
0.5); % Add randomness to the end points, except for the original bound
f_series = f_x(node_series);

plot_x_series = linspace(MIN,MAX,1000); % Different, smoother series just for plotting
plot_f_series = f_x(plot_x_series);

root_guess = NaN(NUM_NODE - 1,1); % Num bracket = num node - 1

for index = 1:(NUM_NODE - 1)
    if f_series(index)*f_series(index+1) < 0 % If the sign of the end points is the
same, fzero will fail.
```

```
        root_guess(index) = fzero(f_x, [node_series(index) node_series(index+1)]);
    end
end

root_guess = root_guess(~isnan(root_guess)) % Using logical indexing to
% eliminate all invalid brackets (NaN root guess = bracket with same sign)
% Note that there will be no duplicated roots, since the brackets does not
% overlap

figure;
plot(plot_x_series, plot_f_series, "DisplayName","Target Function"); hold on
plot(root_guess,zeros(length(root_guess),1),"r*", "DisplayName", "Numerically Located
Roots");
title(sprintf("Numerical Root Finding with fzero() - Bracketed Guess (%d
Intervals)",NUM_NODE -1));
xlabel("x");
ylabel("f(x)");
legend(Location="best");
grid on; grid minor;
```

```
root_guess =

   -0.7435
   -0.6647
   -0.4480
   -0.3310
   -0.1496
   -0.0000
    0.1496
    0.3310
    0.4480
    0.6647
    0.7435
```



Published with MATLAB® R2022a

1B

```
clc; clear; close all;

x_start = 0.5;
x_end = 1.5;
tol = 1e-6;
func = @(x) x*sin(x) - 0.5;

root = myBisection(func,x_start, x_end, tol)

should_be_zero = func(root)
```

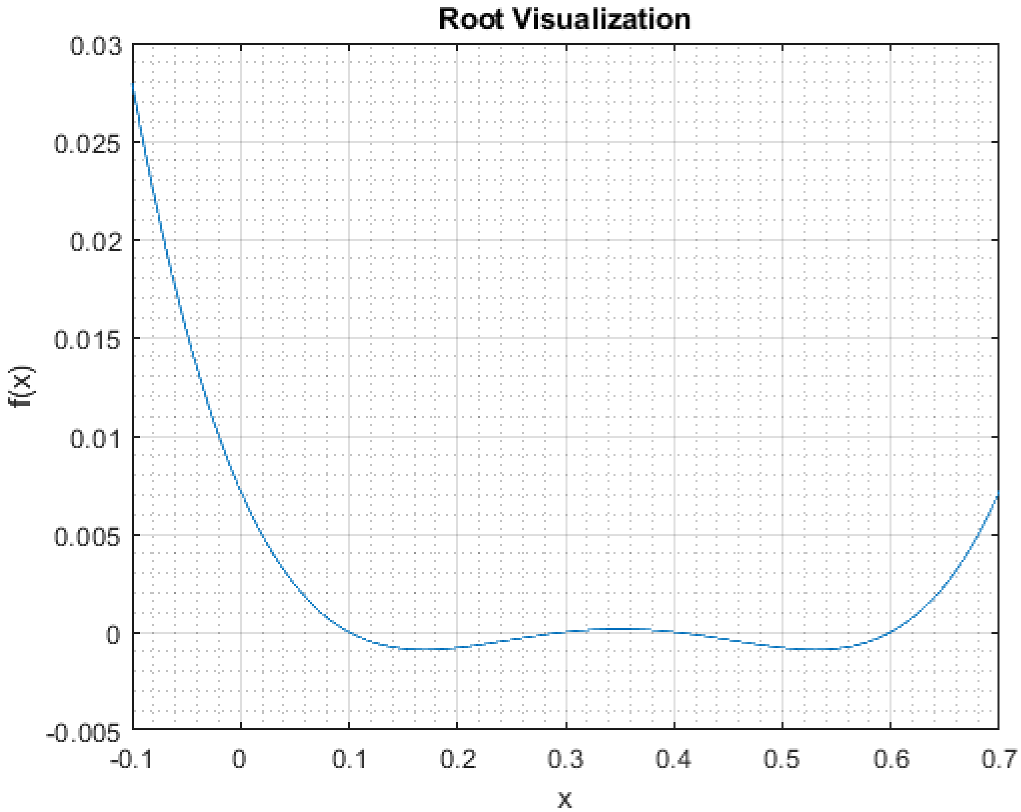
iter_index_list	x_start_list	f_start_list	x_end_list	f_end_list	e_list
0	0.5	0	1.5	0	1
1	0.5	-0.26029	1	0.99624	0.5
2	0.5	-0.26029	0.75	0.34147	0.25
3	0.625	-0.26029	0.75	0.011229	0.125
4	0.6875	-0.13431	0.75	0.011229	0.0625
5	0.71875	-0.063708	0.75	0.011229	0.03125
6	0.73438	-0.026743	0.75	0.011229	0.015625
7	0.73438	-0.0078781	0.74219	0.011229	0.0078125
8	0.73828	-0.0078781	0.74219	0.0016458	0.0039062
9	0.74023	-0.0031237	0.74219	0.0016458	0.0019531
10	0.74023	-0.0007408	0.74121	0.0016458	0.00097656
11	0.74072	-0.0007408	0.74121	0.00045203	0.00048828
12	0.74072	-0.0001445	0.74097	0.00045203	0.00024414
13	0.74072	-0.0001445	0.74084	0.00015373	0.00012207

```
         14              0.74078        -0.0001445       0.74084      4.6071e-06    6.1035e-05
         15              0.74081        -6.995e-05       0.74084      4.6071e-06    3.0518e-05
         16              0.74083       -3.2672e-05       0.74084      4.6071e-06    1.5259e-05
         17              0.74084       -1.4033e-05       0.74084      4.6071e-06    7.6294e-06
         18              0.74084       -4.7127e-06       0.74084      4.6071e-06    3.8147e-06
         19              0.74084       -5.2809e-08       0.74084      4.6071e-06    1.9073e-06
         20              0.74084       -5.2809e-08       0.74084      2.2772e-06    9.5367e-07


root =

    0.7408


should_be_zero =

   5.2968e-07
```

Published with MATLAB® R2022a

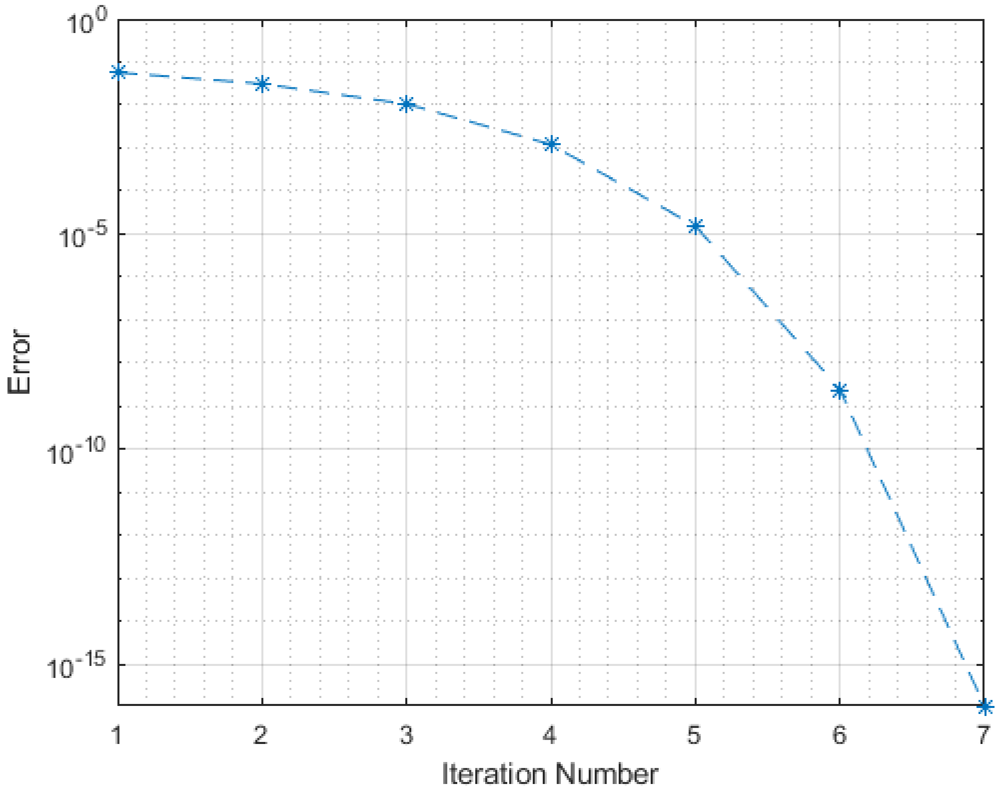
2

```
clc; clear; close all;

f = @(x) x.^4 - 1.4*x.^3 + 0.67*x.^2 - 0.126*x + 0.0072;
df = @(x) 4*x.^3 - 1.4*3*x.^2 + 0.67*2*x - 0.126;

x_plot_series = linspace(-0.1,0.7,1000);
f_plot_series = f(x_plot_series);

figure;
plot(x_plot_series, f_plot_series);
xlabel("x"); ylabel("f(x)");
title("Root Visualization");
grid on; grid minor;

rel_tol = 1e-10;

guess1= 0.0;
guess2= 0.2;
guess3= 0.5;
guess4= 0.7;
```




```
root1 = myNewtonRoot(f,df,guess1,rel_tol)
```

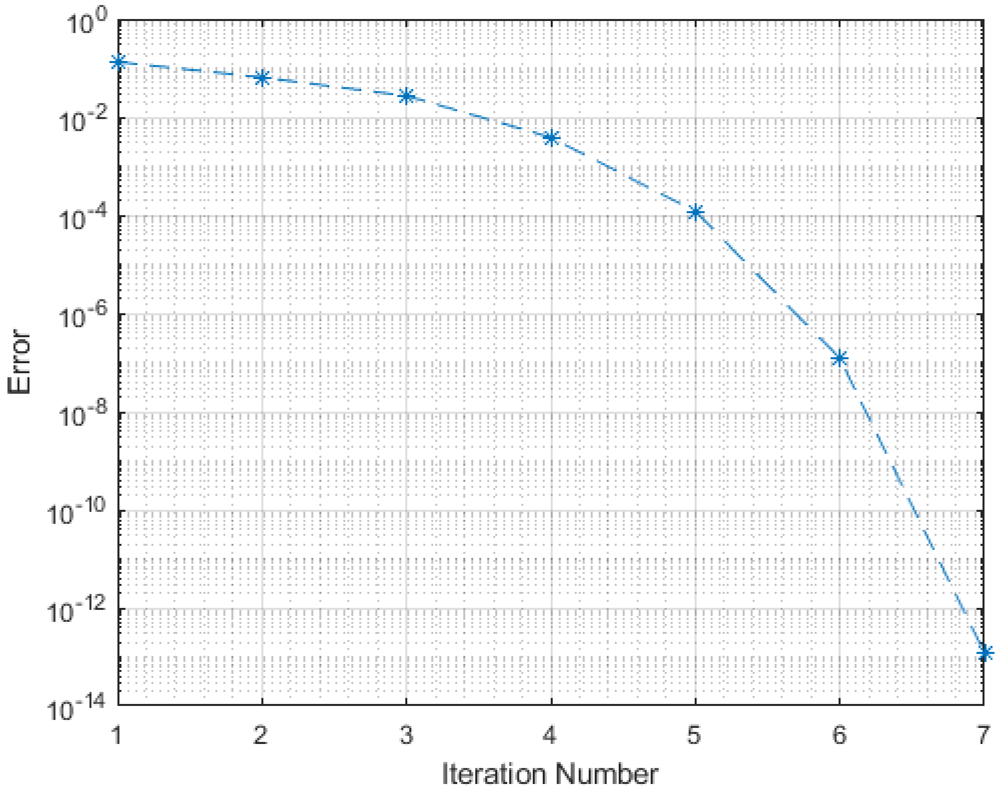
```
    n      x_n          f(x)        df/dx(x)      x_{n+1}      h = |x_{n+1} - x_n|     r = |x_{n+1}/x_n|
    _    ________    __________    _________    ________    ___________________     _________________

    1           0        0.0072       -0.126    0.057143          0.057143                   Inf
    2    0.057143     0.0019372    -0.062397    0.088189          0.031046                1.5433
    3    0.088189    0.00039923    -0.037748    0.098766          0.010576                1.1199
    4    0.098766    3.7506e-05     -0.03077    0.099985         0.0012189                1.0123
    5    0.099985    4.6429e-07     -0.03001         0.1        1.5471e-05                1.0002
    6         0.1     7.421e-11        -0.03         0.1        2.4737e-09                     1
    7         0.1    3.4694e-18        -0.03         0.1        1.1102e-16                     1


root1 =

    0.1000
```




```
root2 = myNewtonRoot(f,df,guess2,rel_tol)
```

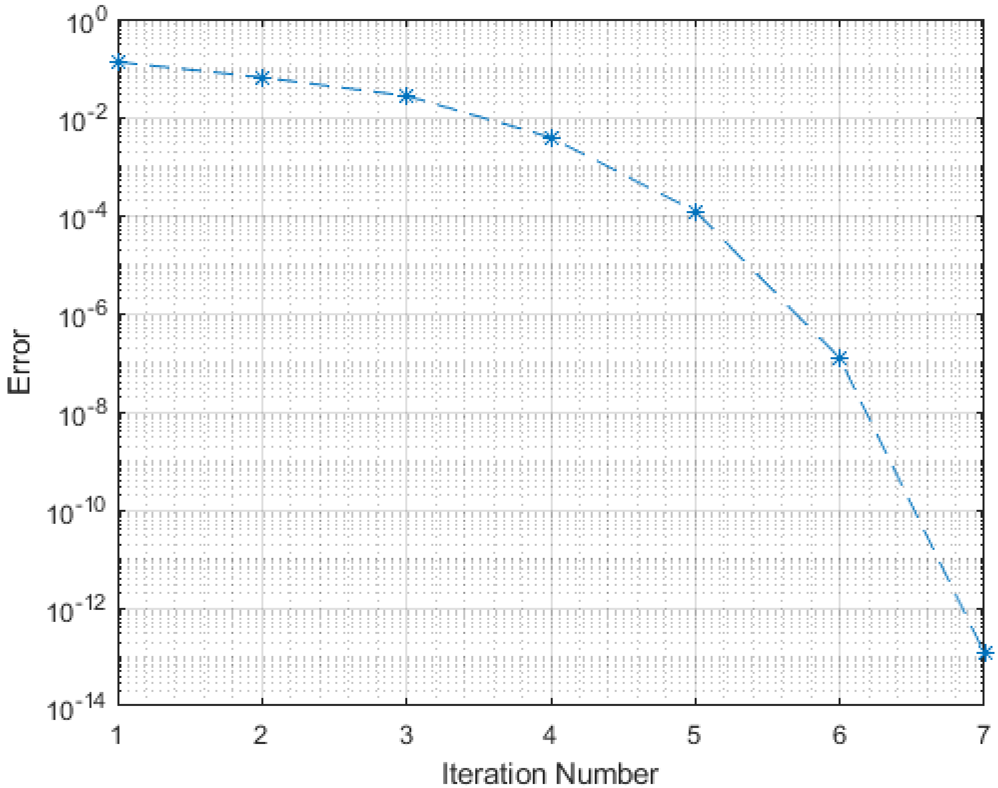
```
    n       x_n          f(x)          df/dx(x)      x_{n+1}     h = |x_{n+1} - x_n|     r = |x_{n+1}/x_n|
    _     _______    ___________    _________     _______     ___________________     _________________

    1         0.2        -0.0008         0.006     0.33333          0.13333               1.6667
    2     0.33333     0.00013827     0.0021481     0.26897         0.064368               0.8069
    3     0.26897    -0.00022746      0.008406     0.29602         0.027059               1.1006
    4     0.29602     -2.463e-05     0.0063878     0.29988        0.0038558                1.013
    5     0.29988    -7.1865e-07      0.006012         0.3       0.00011954               1.0004
    6         0.3    -7.1377e-10         0.006         0.3       1.1896e-07                    1
    7         0.3    -7.1991e-16         0.006         0.3       1.1996e-13                    1
```

```
root2 =

    0.3000
```




```
root3 = myNewtonRoot(f,df,guess3,rel_tol)
```

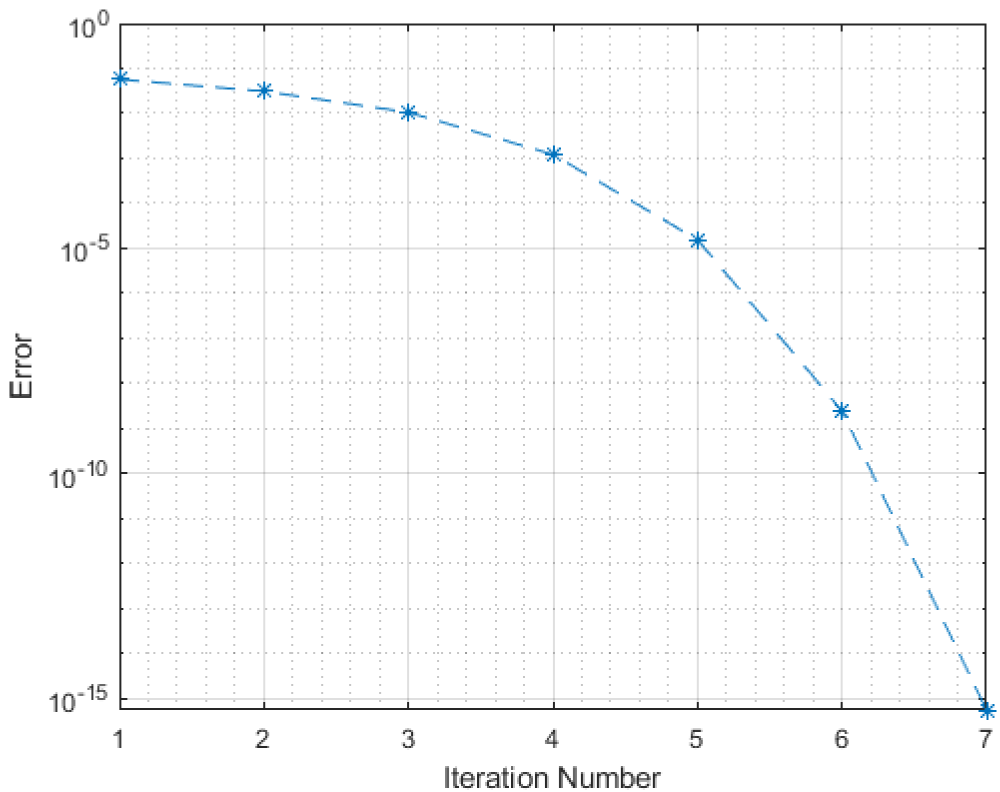
```
    n       x_n          f(x)          df/dx(x)      x_{n+1}      h = |x_{n+1} - x_n|     r = |x_{n+1}/x_n|
    _     _______     ___________     __________     _______     ___________________     _________________

    1         0.5         -0.0008         -0.006     0.36667               0.13333               0.73333
    2     0.36667      0.00013827     -0.0021481     0.43103              0.064368                1.1755
    3     0.43103     -0.00022746      -0.008406     0.40398              0.027059               0.93722
    4     0.40398      -2.463e-05     -0.0063878     0.40012             0.0038558               0.99046
    5     0.40012     -7.1865e-07      -0.006012         0.4            0.00011954                0.9997
    6         0.4     -7.1377e-10         -0.006         0.4            1.1896e-07                     1
    7         0.4     -7.1297e-16         -0.006         0.4            1.1885e-13                     1


root3 =

    0.4000
```




```
root4 = myNewtonRoot(f,df,guess4,rel_tol)
```

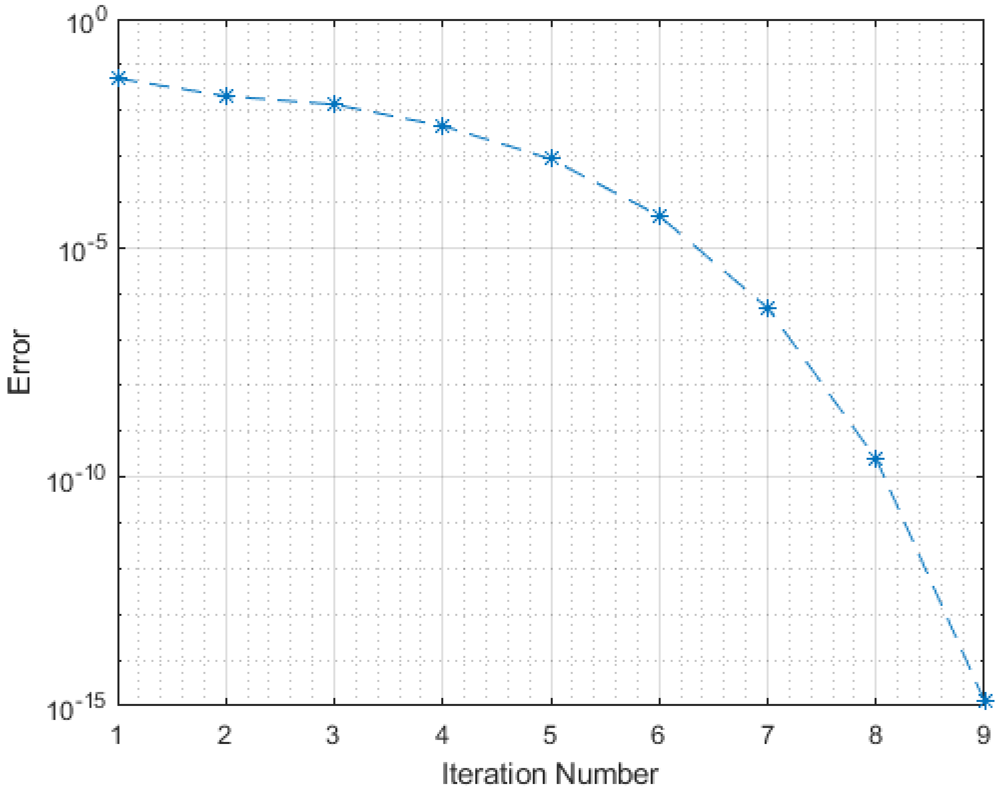
n	x_n	f(x)	df/dx(x)	x_{n+1}	h = |x_{n+1} - x_n|	r = |x_{n+1}/x_n|
1	0.7	0.0072	0.126	0.64286	0.057143	0.91837
2	0.64286	0.0019372	0.062397	0.61181	0.031046	0.95171
3	0.61181	0.00039923	0.037748	0.60123	0.010576	0.98271
4	0.60123	3.7506e-05	0.03077	0.60002	0.0012189	0.99797
5	0.60002	4.6429e-07	0.03001	0.6	1.5471e-05	0.99997
6	0.6	7.421e-11	0.03	0.6	2.4737e-09	1
7	0.6	1.5613e-17	0.03	0.6	5.5511e-16	1

```
root4 =

    0.6000
```




```
guess1_0 = 0.0;
guess1_1 = 0.01;
guess2_0 = 0.2;
guess2_1 = 0.21;
guess3_0 = 0.5;
guess3_1 = 0.49;
guess4_0 = 0.7;
guess4_1 = 0.69;
root1_sec = mySecantRoot(f,guess1_0, guess1_1, rel_tol)
```

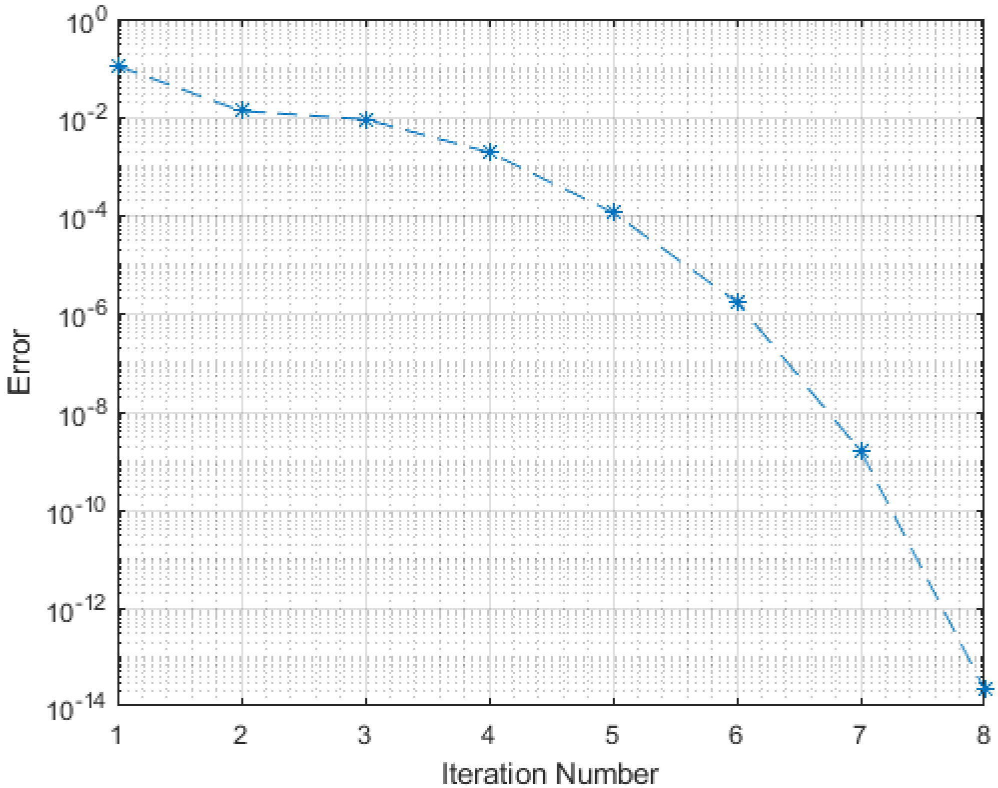
```
    n       x_n         x{n-1}         f(x)        df/dx(x)      x_{n+1}      h = |x_{n+1} - x_n|     r =
|x_{n+1}/x_n|
    _    ________    ________    __________    _________    ________    ___________________
________________

    1           0        0.01     0.0060056     -0.11944    0.060282          0.050282               6.0282
    2        0.01    0.060282     0.0017457     -0.08472    0.080888          0.020606               1.3418
    3    0.060282    0.080888    0.00069372    -0.051054    0.094476          0.013588                1.168
    4    0.080888    0.094476    0.00017536    -0.038148    0.099072         0.0045967               1.0487
    5    0.094476    0.099072    2.8094e-05    -0.032037    0.099949        0.00087693               1.0089
    6    0.099072    0.099949    1.5194e-06    -0.030304         0.1        5.0138e-05               1.0005
    7    0.099949         0.1    1.4455e-08    -0.030016         0.1        4.8157e-07                    1
    8         0.1         0.1    7.5579e-12        -0.03         0.1        2.5193e-10                    1
    9         0.1         0.1    3.8164e-17        -0.03         0.1        1.2768e-15                    1


root1_sec =

    0.1000
```




```
root2_sec = mySecantRoot(f,guess2_0, guess2_1, rel_tol)
```

```
    n       x_n       x{n-1}         f(x)         df/dx(x)      x_{n+1}    h = |x_{n+1} - x_n|    r = |x_{n+1}/x_n|
    _     _______    _______     ___________     _________     _______    ___________________    _________________

    1         0.2       0.21     -0.00073359      0.006641     0.32046                0.11046                1.526
    2        0.21    0.32046      0.00010031      0.007549     0.30718               0.013287              0.95854
    3     0.32046    0.30718      4.0413e-05     0.0045076     0.29821              0.0089656              0.97081
    4     0.30718    0.29821     -1.0893e-05     0.0057225     0.30011              0.0019035               1.0064
    5     0.29821    0.30011      6.8637e-07     0.0060831         0.3             0.00011283              0.99962
    6     0.30011        0.3      1.0034e-08     0.0059942         0.3              1.674e-06              0.99999
    7         0.3        0.3     -9.5885e-12     0.0059999         0.3             1.5981e-09                    1
    8         0.3        0.3      1.3357e-16         0.006         0.3              2.226e-14                    1
```

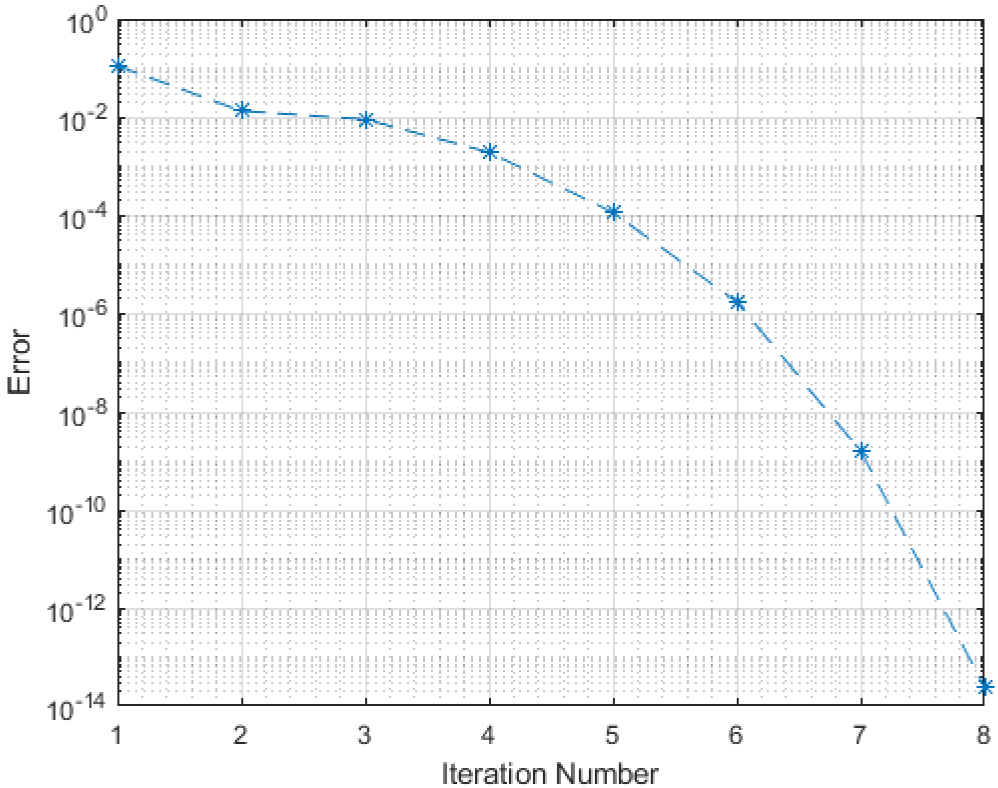
```
root2_sec =

    0.3000
```




```
root3_sec = mySecantRoot(f,guess3_0, guess3_1, rel_tol)
```

n	x_n	x{n-1}	f(x)	df/dx(x)	x_{n+1}	h = |x_{n+1} - x_n|	r = |x_{n+1}/x_n|
_	_______	_______	___________	__________	_______	___________________	_________________
1	0.5	0.49	-0.00073359	-0.006641	0.37954	0.11046	0.77456
2	0.49	0.37954	0.00010031	-0.007549	0.39282	0.013287	1.035
3	0.37954	0.39282	4.0413e-05	-0.0045076	0.40179	0.0089656	1.0228
4	0.39282	0.40179	-1.0893e-05	-0.0057225	0.39989	0.0019035	0.99526
5	0.40179	0.39989	6.8637e-07	-0.0060831	0.4	0.00011283	1.0003
6	0.39989	0.4	1.0034e-08	-0.0059942	0.4	1.674e-06	1
7	0.4	0.4	-9.5886e-12	-0.0059999	0.4	1.5981e-09	1
8	0.4	0.4	1.4745e-16	-0.006	0.4	2.4591e-14	1

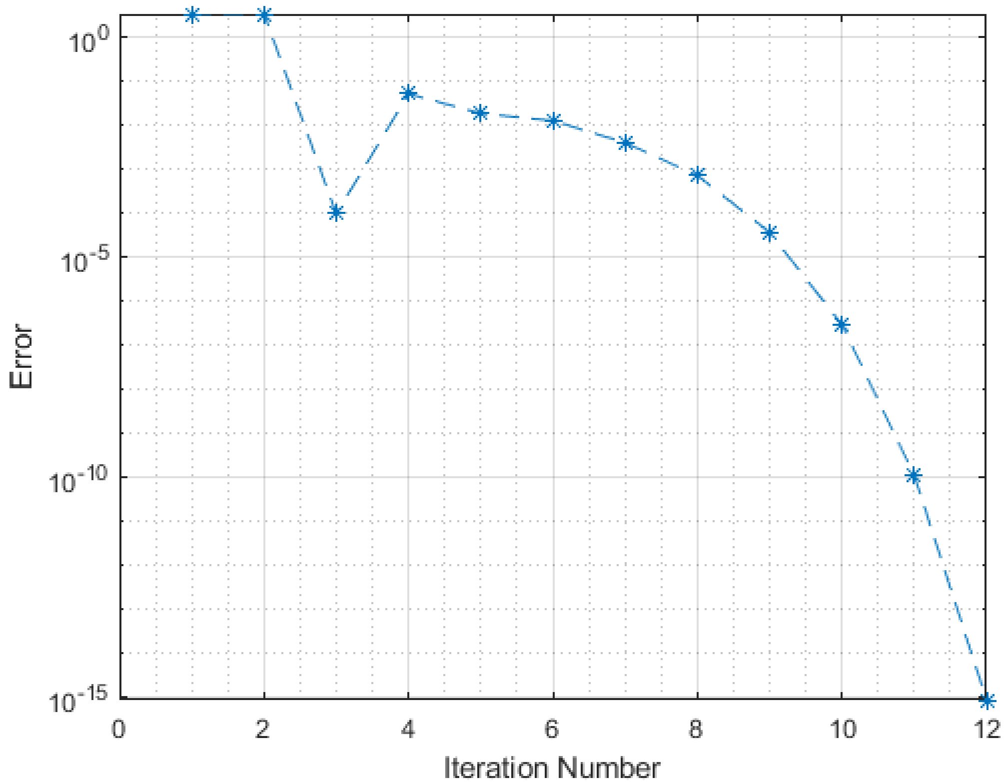
```
root3_sec =

    0.4000
```




```
root4_sec = mySecantRoot(f,guess1_0, guess4_1, rel_tol)
```

n	x_n	x{n-1}	f(x)	df/dx(x)	x_{n+1}	h = |x_{n+1} - x_n|	r = |x_{n+1}/x_n|
1	0	0.69	0.0060056	-0.001731	4.1594	3.4694	6.0282
2	0.69	4.1594	209.65	60.426	0.6899	3.4695	0.16586
3	4.1594	0.6899	0.0059944	60.424	0.6898	9.9205e-05	0.99986
4	0.6899	0.6898	0.0059832	0.11283	0.63677	0.053029	0.92312
5	0.6898	0.63677	0.0015739	0.083149	0.61784	0.018929	0.97027
6	0.63677	0.61784	0.00063979	0.049349	0.60488	0.012965	0.97902
7	0.61784	0.60488	0.00015386	0.037481	0.60077	0.0041051	0.99321
8	0.60488	0.60077	2.34e-05	0.031781	0.60004	0.00073631	0.99877
9	0.60077	0.60004	1.1255e-06	0.030252	0.6	3.7203e-05	0.99994
10	0.60004	0.6	8.9443e-09	0.030012	0.6	2.9803e-07	1
11	0.6	0.6	3.4651e-12	0.03	0.6	1.155e-10	1
12	0.6	0.6	-2.6021e-17	0.03	0.6	8.8818e-16	1

```
root4_sec =

    0.6000
```




```
syms x;
f = x^4 - 1.4*x^3 + 0.67*x^2 - 0.126*x + 0.0072;

MATLAB_symbolic_solver = solve(f, x)
```

```
MATLAB_symbolic_solver =

1/10
3/10
 2/5
 3/5
```

Published with MATLAB® R2022a